HERE

• Claudio “nex” Guarnieri @botherder
  • Security Researcher at Rapid7 Labs
  • Core member of The Shadowserver Foundation
  • Core member of The Honeynet Project
  • Creator of Cuckoo Sandbox
  • Founder of Malwr.com
Mark “rep” Schloesser @repmovsb
- Security Researcher at Rapid7 Labs
- Core Member of The Honeynet Project
- Core developer of Cuckoo Sandbox
- Developed other tools such as Dionaea
HERE

• Jurriaan “skier” Bremer @skier_t
  • Freelance Security Researcher
  • Core developer of Cuckoo Sandbox
NOT HERE

• Alessandro “jekil” Tanasi @jekil
  • Core developer of Cuckoo Sandbox
  • Co-founder of Malwr.com
  • Creator of Hostmap
  • Creator of ImageForensics.org
AGENDA

• Introduction to Sandboxing
• Introduction to Cuckoo
• Customization
• Analysis Internals
• Anti-Sandboxing
• Virtual Machine Introspection
SANDBOXING
How does a sandbox look like?

Software or hardware appliances that receive suspicious files and returns an overview of their functionality.
PROBLEMS

• Process **high volumes**?
• **Automate** specific tasks?
• **Integrate** with defenses?
• Support your T1 **analysts**?
• **Digital forensics**/incident response?
PROS

• Automate the whole analysis process
• Process high volumes of malware
• Usable by virtually anyone
• Get the actual executed code
• Can be very effective if used smartly
CONS

• Can be expensive :-(
• Some portions of the code might not be triggered
• Environment could be detected
• Can be a complete waste
TARGET AUDIENCES

• Incident Handlers
  • Get quick overview of collected samples during an incident response

• Threat Researchers
  • Process high volumes of malware, profile and identify specific families and collect indicators

• Malware Analyst
  • Perform specific analysis operations, e.g. extract configs for banking trojans

• Integration w/ Perimeter Security
  • Automatically process suspicious files from defense layers (e.g. AntiSpam, AntiVirus, DPI ...)
CUCKOO SANDBOX
Automated malware analysis system, easy to use and customize.
Break it!

Wifi: Blackhat
http://10.4.149.79/

Mo Malware?
http://10.4.149.79/bhsamples/
Password: infected
Why?

- We believe in open source
- Empower students and researchers
- Open architecture for more flexibility and creativity
SOME NUMBERS

- Around **50000** lines of code, **Python** and **C**
- More than **2000** commits
- **4** core developers
- ~**25** contributors over time
- ~**15000** downloads in the last 6 months
WHAT YOU NEED TO KNOW

• Basic usage of Linux
• Basic usage of virtual machines
• Knowledge to leverage the results
  • Windows APIs
  • Malicious behaviors
• With Python you can get awesome!
  • Customization
  • Modules
HOW IT WORKS

Pull task
Prepare analysis
Instrument the guest
Execute and log
Process and report
**Key Features**

- Almost *everything is a module*
- Completely automated
- Run *concurrent* analysis
- Able to *trace* processes recursively
- *Customize analysis* process
- Create behavioral *signatures*
- Customize processing and reporting
DEMO
First look at Cuckoo Sandbox
GETTING STARTED
**Requirements and Expectations**

- **What is your goal?**
- **Who** is going to use the sandbox?
- How are they going to **consume the data**?
- **How many samples** do you expect?
- What kind of results are **mostly relevant**?
- Do you need **all features** to meet your goal?
DESIGN YOUR ENVIRONMENT

• Do you want to run **Office** exploits?
• Do you want to run **PDF** exploits?
• Do you want to run **64 bit** malware?
• Do you want to run **URLs**?
• Do you need **script interpreters**?
IDEAS

• Look for the most exploitable version of applications (metasploit, exploitdb, etc.)
  • http://www.rapid7.com/db/

• Create multiple VMs with multiple versions of applications

• Leave some fake credentials and tokens around

• Disguise the VM as much as possible
**Deployment Examples**

1. **Local support tool for case analysis**
   - Air-gapped, fake services (e.g. DNS, HTTP), multiple configurations

2. **Large scale** feed processing
   - Multiple instances, full Internet, automatic digesting of results (extracting indicators)

3. **On-Demand** Public or Internal service
   - Full Internet, separate network
INSTALLATION IN A NUTSHELL

- Install **VirtualBox**, VMWare or QEMU/KVM
- Download & extract **Cuckoo**
- Install **dependencies**
- Create a virtual machine, copy over and run `agent.py` and take a snapshot *(need to be able to communicate with the host)*.
- Configure the files in **conf/**
- `$ python cuckoo.py`
SETUP DISCLAIMERS

• It’s not point-and-click, you need to work a bit
• Virtualization software are not intended for massive and continuous restore
• There are some key steps to do, if one is skipped nothing works
• There’s an extensive documentation, mailing list and Q&A platform: check them out.
Submission

- `utils/submit.py`
- `utils/api.py`
- Django Web Interface
- Python API

```python
1 import sys
2 sys.path.append('/opt/cuckoo/')
3 from lib.cuckoo.core.database import Database
4
5 db = Database()
6 db.add_path(file_path)
7 db.add_url(url)
```
OPTIONS

• Analysis Package + Options
• Timeout
• Priority
• Machine
• Platform
• Memory Dump
• Enforce Timeout
• Clock
• Tags
RESULTS

- Raw results stored in `storage/analysis/<id>/`
- Reports stored in `storage/analysis/<id>/reports/`
  - Depends on what was enabled in `conf/reporting.conf`
RESULTS

- Trace of **API calls**
- File dumps
- Screenshots
- **Network traffic**
- Process memory dump
- System memory dump
COMPONENTS
COMPONENTS

• **Core**
  • Main service running on the host. Handles analysis machines, digest submissions and process results.

• **Analyzer**
  • Runs inside the analysis machine. Runs the malware, instruments it and generate results, streamed to the Core.
  • Selected according to the platform of the machine, currently only Windows.
CORE MODULES
**Machinery Modules**

- In **Core** (under `modules/machinery/`)
- Python class
- **Define interaction with the virtualization software**
- Default:
  - VirtualBox
  - VMWare
  - QEMU/KVM
  - Generic LibVirt
# Copyright (C) 2010-2013 Cuckoo Sandbox Developers.
# This file is part of Cuckoo Sandbox - http://www.cuckoosandbox.org
# See the file 'docs/LICENSE' for copying permission.

import logging

from lib.cuckoo.common.abstracts import LibVirtMachinery

class KVM(LibVirtMachinery):
    """Virtualization layer for KVM based on python-libvirt."""

    # Set KVM connection string.
    dsn = "qemu:///system"
Auxiliary Modules

- In **Core** (under modules/auxiliary/)
- Python class
- No specific use, just run concurrently to each analysis.
- Default:
  - Network traffic capture
PROCESSING MODULES

• In Core (under modules/processing/)
• Python class
• Process raw results (sample, API logs, files, memory)
• Populate collection of results
DEFAULT PROCESSING MODULES

- Analysis Info
- File Hashes & Yara Signatures
- Behavior Analysis
- Dropped Files
- Volatility Memory Analysis
- PCAP Analysis
- PE32 Static Analysis
- Strings
- VirusTotal Results
```
signatures = {
    'namespace': 'rule pivars {strings: $a = { 
53 74 75 62 50 61 74 68 ?? 53 4F 46 54 57 41 52
45 5C 43 6C 61 73 73 65 73 5C 68 74 74 70 5C 73
68 65 6C 6C 5C 6F 70 65 6E 5C 63 6F 6D 66 6D 61 6E
64 [22] 53 6F 66 74 77 61 72 65 5C 4D 69 63 72 6F
73 6F 66 74 5C 41 63 74 69 76 65 20 53 65 74 75
70 5C 49 6E 73 74 61 6C 6C 5C 65 64 20 43 6F 6D 70
6F 6E 65 6E 74 73 5C } condition: $a}'}

class PoisonIvy(Processing):
    def run(self):
        self.key = "poisonivy"
        results = {}

        rules = yara.compile(sources=signatures)

        dumps = []
        for root, dirs, files in os.walk(self.pmemory_path):
            if files:
                for file_name in files:
                    dumps.append(os.path.join(root, file_name))

        for dump in dumps:
            matches = rules.match(dump)

            if not matches:
                continue

            data = open(dump, "rb")

            offset = matches[0].strings[0][0]
            data.seek(offset + 0x6eb)
            results["identifier"] = data.read(100).split("\00")[0]
            data.seek(offset + 0x2a2)
            results["persistence"] = data.read(100).split("\00")[0]
            data.seek(offset - 0x27e)
            results["server"] = data.read(100).split("\00")[0]

            break

        return results
```
DEMO
Running PoisonIvy
SIGNATURES

• In **Core** (under `analyzer/windows/modules/signatures/`)

• Python class

• Isolate specific events
  • Identify malware family
  • Identify malicious behavior
  • Extract configuration
  • ...

...
from lib.cuckoo.common.abstracts import Signature

class Prinimalka(Signature):
    name = "banker_prinimalka"
    description = "Detected Prinimalka banking trojan"
    severity = 3
    categories = ["banker"]
    families = ["prinimalka"]
    authors = ["nex"]
    minimum = "1.0"
    evented = True

def event_apicall(self, call, process):
    if call["api"].startswith("RegSetValueEx"):
        if self.get_argument(call, "ValueName").endswith("_opt_server1"):
            server = self.get_argument(call, "Buffer").rstrip("\\x00")
            self.description += " (C&C: {})".format(server)
        return True
REPORT
Prinimalka Report
COMMUNITY SIGNATURES

• Community Repository
  • https://github.com/cuckoobox/community

• utils/community.py –signatures (--force)
SHARING IS CARING!
REPORTING MODULES

- In **Core** (under `analyzer/windows/modules/reporting/`)
- Python class
- Make use of abstracted results
- Default:
  - JSON
  - HTML
  - MAEC
  - MongoDB
import os
import json
import codecs

from lib.cuckoo.common.abstractions import Report
from lib.cuckoo.common.exceptions import CuckooReportError

class JsonDump(Report):
    """Saves analysis results in JSON format."""

def run(self, results):
    """Writes report.
    @param results: Cuckoo results dict.
    @raise CuckooReportError: if fails to write report.
    """
    try:
        report = codecs.open(os.path.join(self.reports_path, "report.json"), "w", "utf-8")
        json.dump(results, report, sort_keys=False, indent=4)
        report.close()
    except (UnicodeError, TypeError, IOError) as e:
        raise CuckooReportError("Failed to generate JSON report: %s" % e)
ANALYZER MODULES
ANALYSIS PACKAGES

• In Analyzer (under analyzer/windows/modules/packages/)
• Python modules
• Define how to interact with the malware and the system
• Can be used for scripting tasks
from lib.common.abstractions import Package
from lib.api.process import Process
from lib.common.exceptions import CuckooPackageError

class Browser(Package):
    def start(self, path):
        free = self.options.get("free", False)
        args = self.options.get("arguments", None)
        suspended = True
        if free:
            suspended = False

        b = Process()
        b.execute(path="C:\\Program Files\\Internet Explorer\\iexplore.exe", suspended=False)
        b.close()

        p = Process()
        if not p.execute(path=path, args=args, suspended=suspended):
            raise CuckooPackageError("Unable to execute initial process, analysis aborted")

        pid = None
        if not free and suspended:
            p.inject()
            p.resume()
            p.close()
            pid = p.pid

        return pid

    def check(self):
        return True

    def finish(self):
        if self.options.get("procmemdump", False):
            for pid in self.pids:
                p = Process(pid=pid)
                p.dump_memory()

        return True
REPORT

Sikypot report example
AUXILIARY MODULES

- In **Analyzer** (under `analyzer/windows/modules/auxiliaries/`)
- Python modules
- Run concurrently to the analysis
- Default:
  - Screenshots
  - Emulation of human interaction
CUCKOOMON
Cuckoomon

- Instrumentation based on DLL injection
- DLL employs inline function hooking
- Logging API calls + parameters to the host over a TCP connection
API HOOKING OVERVIEW

- **CuckooMon** logs about 170 APIs
  - Process & Thread Management
  - File Manipulation
  - Registry
  - Network
  - Windows GUI
  - Windows Services
  - Miscellaneous

- **Adding new hooks** where needed
- **Special handling of certain API calls**
  - Dumping created and modified files
  - “Follow” child processes & injected processes
CHILD PROCESS INJECTION

- We’re interested in Child Processes
  - It’s part of what the malware does
- Child processes created using **CreateProcess**
- When called, we
  - Create the process in **suspended** mode
  - Inject Cuckoomon & set all hooks, etc.
  - **Resume** the main thread
**Injected Process Injection**

- Malware often injects into e.g. `explorer.exe`
  - E.g., avoids malicious processes in the process list
- Various injection methods, such as `CreateRemoteThread`, `QueueUserAPC`, etc.
- When such injection is detected, we
  - Inject Cuckoomon first
  - Continue with the malware process injection
• Hook low-level & higher level functions
  • Low-level functions give more information
  • High-level functions wrap around them
• E.g., `system()` & `CreateProcess()`
  • `system()` takes a string, e.g., “echo hello | ./world”
  • Might execute multiple processes, pipes, etc.
  • We log `system()` with the string
  • We inject each process created by `CreateProcess()`
EXAMPLE API HOOK

HOOKDEF(BOOL, WINAPI, CreateProcessInternalW,
    __in_opt    LPVOID lpUnknown1,
    __in_opt    LPWSTR lpApplicationName,
    __inout_opt LPWSTR lpCommandLine,
    __in_opt    LPSECURITY_ATTRIBUTES lpProcessAttributes,
    __in_opt    LPSECURITY_ATTRIBUTES lpThreadAttributes,
    __in        BOOL bInheritHandles,
    __in        DWORD dwCreationFlags,
    __in_opt    LPVOID lpEnvironment,
    __in_opt    LPWSTR lpCurrentDirectory,
    __in        LPSTARTUPINFO lpStartupInfo,
    __out       LPPROCESS_INFORMATION lpProcessInformation,
    __in_opt    LPVOID lpUnknown2
) {
    [...]  
    BOOL ret = Old_CreateProcessInternalW(lpUnknown1, lpApplicationName,
    lpCommandLine, lpProcessAttributes, lpThreadAttributes,
    bInheritHandles, dwCreationFlags, lpEnvironment, lpCurrentDirectory,
    lpStartupInfo, lpProcessInformation, lpUnknown2);
    [...]  
}
MAGIC HOOKING ENGINE ™

• Avoids **hook recursion** for logging
  • One `system()` may call many `CreateProcess()`’s
  • Log only the `system()`
• Automatically restores the **Last Error** code
  • If an API failed, but logging was successful
  • Does that make the Last Error successful?!
**StubDLL (WIP)**

- Don’t employ inline hooking directly
- Create a **Shadow DLL**
  - Copy the original DLL in-memory
  - Add slightly more stealthy hooks
- Avoids trivial **hook detection**
  - `if(*function_address == 0xe9) { /* hook found */ }`
RETURN ADDRESS CHECKING (WIP)

- When injected into e.g. `explorer.exe`
- CuckooMon also logs explorer’s function calls
- Retaddr Checking tries to avoid unrelated data
  - Before logging, check ret addrs in the stacktrace
  - Anything *interesting*? Then log.
    - Injected memory, overwritten memory, etc.
    - Otherwise ignore this function call
- Drawback: not always useful
ANTI-SANDBOXING
With **sandboxes** getting popular, malware writers are increasingly **trying to** bypass them.
ANTI-SLEEP

• Cuckoo Sandbox skips sleeps that are launched within the first seconds of a process execution.
HOOKDEF(NTSTATUS, WINAPI, NtDelayExecution,
    _in_ BOOLEAN Alertable,
    _in_ PLARGE_INTEGER DelayInterval
) {
    NTSTATUS ret = 0;

    // do we want to skip this sleep?
    if(sleep_skip_active != 0) {
        FILETIME ft; LARGE_INTEGER li;
        GetSystemTimeAsFileTime(&ft);
        li.HighPart = ft.dwHighDateTime;
        li.LowPart = ft.dwLowDateTime;

        // check if we're still within the hardcoded limit
        if(li.QuadPart < time_start.QuadPart + MAX_SLEEP_SKIP_DIFF * 10000) {
            time_skipped.QuadPart += -DelayInterval->QuadPart;

            // notify skipped how much we've skipped
            unsigned long milli = -DelayInterval->QuadPart / 10000;
            LOQ("ls", "Milliseconds", milli, "Status", "Skipped");
            return ret;
        }
    }
    else {
        sleep_skip_active = 0;
    }

    LOQ("l", "Milliseconds", -DelayInterval->QuadPart / 10000);
    return Old_NtDelayExecution(Alertable, DelayInterval);
}
MOUSE-MONITOR

- Malware can observe for:
  - Movements of the cursor
  - Clicks of the mouse buttons
- They don’t start until such events are observed.
- Good example is **Upclicker**.
ANTI-MOUSE-MONITOR

• Cuckoo Sandbox *emulates human interaction*
  • Move the mouse cursor
  • Click on mouse buttons
  • Click on dialogs
```python
class Human(Auxiliary, Thread):
    """""""Human after all""
    
    def __init__(self):
        Thread.__init__(self)
        self.do_run = True

    def stop(self):
        self.do_run = False

    def run(self):
        while self.do_run:
            move_mouse()
            click_mouse()
            USER32.EnumWindows(EnumWindowsProc(foreach_window), 0)
            KERNEL32.Sleep(1000)
```
<table>
<thead>
<tr>
<th>TIME</th>
<th>API</th>
<th>ARGUMENTS</th>
<th>STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>2013-04-09 22:13:33,534</td>
<td>SetWindowsHookExA</td>
<td>HookIdentifier: 14, ProcedureAddress: 0x004010b0, ModuleAddress: 0x00400000, ThreadId: 0</td>
<td>success</td>
</tr>
<tr>
<td>2013-04-09 22:13:33,874</td>
<td>UnhookWindowsHookEx</td>
<td>HookHandle: 0x008400a9</td>
<td>success</td>
</tr>
<tr>
<td>2013-04-09 22:13:33,874</td>
<td>LdrLoadDll</td>
<td>Flags: 1240168, FileName: advapi32, BaseAddress: 0x77dd0000</td>
<td>success</td>
</tr>
<tr>
<td>2013-04-09 22:13:33,874</td>
<td>LdrLoadDll</td>
<td>Flags: 1240168, FileName: ntdll, BaseAddress: 0x7c900000</td>
<td>success</td>
</tr>
<tr>
<td>2013-04-09 22:13:33,874</td>
<td>LdrLoadDll</td>
<td>Flags: 1240168, FileName: user32, BaseAddress: 0x7e410000</td>
<td>success</td>
</tr>
<tr>
<td>2013-04-09 22:13:33,904</td>
<td>LdrLoadDll</td>
<td>Flags: 1240168, FileName: advpack, BaseAddress: 0x75260000</td>
<td>success</td>
</tr>
<tr>
<td>2013-04-09 22:13:33,904</td>
<td>LdrLoadDll</td>
<td>Flags: 1240076, FileName: advapi32.dll, BaseAddress: 0x77dd0000</td>
<td>success</td>
</tr>
<tr>
<td>2013-04-09 22:13:33,904</td>
<td>LdrGetProcedureAddress</td>
<td>ModuleHandle: 0x77dd0000, FunctionName: CheckTokenMembership, Ordinal: 0</td>
<td>success</td>
</tr>
<tr>
<td>TIME</td>
<td>API</td>
<td>ARGUMENTS</td>
<td>STATUS</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------------------</td>
<td>---------------------------------------------------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>2013-04-09 22:13:33,534</td>
<td>SetWindowsHookExA</td>
<td>HookIdentifier: 14 ProcedureAddress: 0x004010b0 ModuleAddress: 0x00400000 ThreadId: 0</td>
<td>success</td>
</tr>
<tr>
<td>2013-04-09 22:13:33,874</td>
<td>UnhookWindowsHookEx</td>
<td>HookHandle: 0x008400a9</td>
<td>success</td>
</tr>
<tr>
<td>2013-04-09 22:13:33,874</td>
<td>LdrLoadDll</td>
<td>Flags: 1240168 FileNm: advapi32.dll BaseAddress: 0x75260000</td>
<td>success</td>
</tr>
<tr>
<td>2013-04-09 22:13:33,904</td>
<td>SetWindowsHookExA</td>
<td>HookIdentifier: 14 ProcedureAddress: 0x004010b0 ModuleAddress: 0x00400000 ThreadId: 0</td>
<td></td>
</tr>
<tr>
<td>2013-04-09 22:13:33,904</td>
<td>UnhookWindowsHookEx</td>
<td>HookHandle: 0x008400a9</td>
<td>success</td>
</tr>
<tr>
<td>2013-04-09 22:13:33,904</td>
<td>LdrGetProcedureAddress</td>
<td>ModuleHandle: 0x77dd0000 FunctionName: CheckTokenMembership Ordinal: 0</td>
<td></td>
</tr>
</tbody>
</table>
```python
from lib.cuckoo.common.abstracts import Signature

class HookMouse(Signature):
    name = "antisandbox_mouse_hook"
    description = "Installs an hook procedure to monitor for mouse events"
    severity = 3
    categories = ["hooking", "anti-sandbox"]
    authors = ["nex"]
    minimum = "1.0"
    evented = True

def event_apicall(self, call, process):
    if not call["api"].startswith("SetWindowsHookEx"):
        return

    if int(self.get_argument(call, "HookIdentifier")) in [7, 14]:
        if int(self.get_argument(call, "ThreadId")) == 0:
            return True
```
ANTI-VIRTUALIZATION

• It’s painful
• **Depends** on the virtualization software of your choice
• You can **do something** about it
• However you **won’t be able to kill all indicators**
ANTI-VIRTUALIZATION TRICKS

• Find **processes**: VBoxService.exe, vmtoolsd.exe
• Find **files or devices**: \Device\VBoxMouse
• Detect **available libraries**:
  LoadLibrary(‘VBoxOGL.dll’)
• Detect **BIOS version**
• Detect **disk description**:
  IOCTL_STORAGE_QUERY_PROPERTY, IOCTL_SCSI_MINIPORT
• Detect **disk size**: IOCTL_DISK_GET_DRIVE_GEOMETRY
• Detect **guest tools**
• Find **windows**: FindWindow(‘VBoxTrayToolWnd’)
### NtCreateFile

- **FileHandle**: 0x00000094
- **DesiredAccess**: 0xc0100080
- **FileName**: Scsi0
- **CreateDisposition**: 1
- **ShareAccess**: 3

**success**

### DeviceIoControl

- **DeviceHandle**: 0x00000094
- **IoControlCode**: 315400
- **InBuffer**:
  - x1c\x00\x00\x00\x00SCSIDISK\x02\x00\x00\x00\x01\x05\x1b\x00\x00\x00\x01\x02\x00
  - ... (truncated)
- **OutBuffer**: 0x1c\x00\x00\x00\x00SCSIDISK\x02\x00\x00\x00\x01\x05\x1b\x00\x00\x00\x01\x02\x00

**success**
```
NtCreateFile
HANDLE FileHandle = 0x000000094
DesiredAccess = 0xc0100080
FileName = \Scsi0:
CreateDisposition = 1
ShareAccess = 3

DeviceIoControl
DeviceHandle = 0x000000094
IoControlCode = 315400
InBuffer: 0x1c 0x00 0x00 0x00SCSIDISK 0x02 0x00 0x00 0x00
  0x01 0x05 0x1b 0x00 0x00 0x00 0x00 0x02 0x00
  0x00 0x00 0x02 0x00 0x00 0x00 0x00 0x00
  0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
  0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
  0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
  0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
  0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
  0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
  0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
  0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00

success
```

from lib.cuckoo.common.abstracts import Signature

class DiskInformation(Signature):
    name = "antivm_generic_disk"
    description = "Queries information on disks, possibly for anti-virtualization"

    [...]

def event_apicall(self, call, process):
    indicators = [
        "scsi0",
        "physicaldrive0"
    ]

    ioctlsl = [
        "2954240", # IOCTL_STORAGE_QUERY_PROPERTY
        "458752", # IOCTL_DISK_GET_DRIVE_GEOMETRY
        "315400" # IOCTL_SCSI_MINIPORT
    ]

    if process is not self.lastprocess:
        self.handle = None
        self.lastprocess = process

    if not self.handle:
        if call["api"] == "NtCreateFile":
            file_name = self.get_argument(call, "FileName")
            for indicator in indicators:
                if indicator in file_name.lower():
                    self.handle = self.get_argument(call, "FileHandle")
        else:
            if call["api"] == "DeviceIoControl":
                if self.get_argument(call, "DeviceHandle") == self.handle:
                    if str(self.get_argument(call, "IoControlCode")) in ioctlsl:
                        return True
VIRTUALBOX EXTRA DATA

$ VBoxManage setextradata <label> VBoxInternal/Devices/ +

- pcbios/0/Config/DmiBIOSFirmwareMajor
- pcbios/0/Config/DmiBIOSFirmwareMinor
- pcbios/0/Config/DmiBIOSReleaseDate
- pcbios/0/Config/DmiBIOSReleaseMajor
- pcbios/0/Config/DmiBIOSReleaseMinor
- pcbios/0/Config/DmiBIOSVendor
- pcbios/0/Config/DmiBIOSVersion
- pcbios/0/Config/DmiChassisAssetTag
- pcbios/0/Config/DmiChassisSerial
- pcbios/0/Config/DmiChassisVendor
- pcbios/0/Config/DmiSystemFamily
- pcbios/0/Config/DmiSystemProduct
- pcbios/0/Config/DmiSystemSKU
- pcbios/0/Config/DmiSystemSerial
- pcbios/0/Config/DmiSystemUuid
- pcbios/0/Config/DmiSystemVendor
- pcbios/0/Config/DmiSystemVersion
- piix3ide/0/Config/Port0/ATAPIProductId
- piix3ide/0/Config/Port0/ATAPIRevision
- piix3ide/0/Config/Port0/ATAPIVendorId
- piix3ide/0/Config/PrimaryMaster/FirmwareRevision
- piix3ide/0/Config/PrimaryMaster/ModelNumber
- piix3ide/0/Config/PrimaryMaster/SerialNumber
DO NOT INSTALL THE GUEST ADDITIONS.
**Windows Registry**

- HKLM\HARDWARE\Description\System\SystemBiosVersion
- HKLM\HARDWARE\Description\System\VideoBiosVersion
- HKLM\HARDWARE\DEVICEMAP\Scsi\Scsi Port 0\Scsi Bus 0\Target Id 0\Logical Unit Id 0
- HKLM\SYSTEM\CurrentControlSet\Enum\IDE
Want to find samples?

- http://malwr.com/analysis/search/
- signature:virtualization
WHERE WE FALL SHORT

• In order to monitor API calls we currently use **inline hooks**
• That requires replacing the first bytes of a function to **hijack it to a different location**
• These trampolines can be **easily detected**
Anti-Malwr + (Cuckoo)
ALTERNATIVE ANALYSIS TECHNIQUES

• CuckooMon: userland DLL injection
  • comfortable, simple, still effective
  • sadly easy to detect/circumvent
• Commercial sandboxes often kernel based tracing, sometimes combined with userland components
• Even harder to detect: introspection from outside the OS

Cuckoo VMI?
GENERALIZING CUCKOO LOG DATA

• Necessary changes to Cuckoo
  • Generalizing behavior semantics for Mac/Linux platforms anyway

• More visibility / possibilities with VMI
  • Might need more flexible configuration of the analyzer engine
VIRTUAL MACHINE INTROSPECTION

• Observe the memory and execution flow from the outside
• Look at kernel structures to differentiate between processes / libraries
• Depending on virtualization technique use its features to pause VM execution and extract function arguments / memory contents
What do we need for \textit{inspecting Windows from the outside}?

- Processes (track cr3)
- Libraries / Modules

Kernel structures:
- EPROCESS (ActiveProcessHead list)
- Process Object Tables (HANDLE_TABLE)
- Virtual Address Descriptor tree (VAD tree)
WIP: CuckooVMI based on QEMU

- QEMU: binary translation engine: TCG (Tiny Code Generator)
- Great base for both coarse- and fine-grained tracing of the guest and its processes
- Focus on Windows XP/7 – find kernel process structs and track their executable memory
- Full tracing or specific locations
- Never miss executed code
AUTOMATED FUNCTIONCALL LOGGING

- Windows APIs mostly use stdcall calling convention
  - Callee cleans up the stack, EAX = returnvalue
- This allows for generic parameter logging
  - Note stack pointer when entering function
  - Note stack pointer when returning
  - Everything in between was a parameter
- Still needs knowledge of types for special logging (Strings, structs, etc)
AUTOMATED LOGGING CONT.

- Type information can be automatically extracted from development headers

```c
NTSTATUS NtCreateFile(
HANDLE* FileHandle,
FILE_ACCESS_MASK DesiredAccess,
OBJECT_ATTRIBUTES* ObjectAttributes, IO_STATUS_BLOCK* IoStatusBlock, LARGE_INTEGER*, ...)
```

- Specify list of interesting variables in all those structs, generate dereference/offset code automatically

- Comes down to only implementing specific **code for elementary types** (char *, wchar_t *, UNICODE_STRING)
DEMO
Virtual Machine Introspection
**RELATED WORK: DECAF PLATFORM**

- Qemu based analysis framework out of Berkeley
- Base of Android analysis project “**DroidScope**”
- Also supports tracing / analysing x86 Windows guests
- Parts from **BitBlaze/TEMU** and other related projects
- Rich hooking API
  - Specific addresses, all basic blocks, memory write, etc
- Experimental taint tracking features
- **Too many features and too invasive** (outdated QEMU, etc) for our purpose
ALTERNATIVE VMI SOLUTIONS

• Thin hypervisor for VM performance
  • Use page protection faults to trap to the hypervisor at interesting locations

• Other rootkit techniques? UEFI drivers?

• Cuckoo hopefully grows to other platforms and several analyzer techniques to choose from
  • Brings even more customization / flexibility
CONCLUSIONS
SUMMING UP

• **Open source** solution (and will remain so)
• Flexible and customizable
• Easy to integrate
• Very actively developed
**FUTURE**

- Create more and more *signatures*!
- Improve *performances*
- Continue work on *VMI* techniques
- **Bare-metal** support (almost done)
- Add *Linux* support
- Add *Mac OS X* support
- Feedback?
OTHER STUFF

• Malwr
  • https://malwr.com

• VxCage
  • https://github.com/cuckoobox/vxcage